Priprava na ustni izpit iz ARS1

1) Von Neumannov računalniški model

Stroj mora izpolnjevati naslednje pogoje:

- sestavljajo ga trije osnovni deli: CPE, glavni pomnilnik in V/I sistem.
- ima program, shranjen v glavnem pomnilniku, ki vodi delovanje stroja
- CPE jemlje ukaze iz glavnega pomnilnika, kamor kaže PC – strojni ukazi (FETCH) in jih izvršuje enega za drugim in povečuje PC – register, ki vsebuje naslov naslednjega ukaza (EXECUTE)

CPE – skoraj vse dogajanje se odvija pod njeno kontrolo. Razdelimo jo lahko na kontrolno enoto in podatkovno enoto (aritmetično logična + registri). Registri so programsko dostopni in programsko nedostopni(niso nujni). Omogočajo hitrejše delovanje.
GLAVNI POMNILNIK – v njem so ukazi in operandi, ki jih uporablja CPE. Sestavljen je iz več pomnilniških besed, katerih vsaka ima enoličen naslov. Pri tem modelu računalnik ne loči med ukazi in operandi.
V/I sistem – je namenjen prenosu informacij v/iz zunanjega sveta. Sestavljen je iz V/I naprav. Nekatere izmed njih služijo tudi kot pomožni pomnilnik. Fizično največji del računalnika.

Delovanje računalnika popolnoma določajo ukazi, v dveh korakih: fetch in execute. PC – PC +1. Do izjeme pride le pri prekinitvah, pasteh in skokih. PC – PC + n.

Obstajati mora mehanizem, ki ukaze naloži v glavni pomnilnik – bootstrap loader.

Von Neumannov model je ukazno pretokovni, obstaja še podatkovno pretokovni.

2) DMA krmilnik, V/I procesorji

Vsaka V/I naprava je priključena preko krmilnika, ki omogoča prenos iz/v njo. Videti je kot določeno število registrov, s tem da branje ali pisanje v njih sproži določeno akcijo v napravi.

DMA (Direct Memmory Access) označuje neposredno komunikacijo med glavnim pomnilnikom in V/I sistemom (nasprotje od programskega V/I, ko z V/I napravo komunicira le CPE – računalnik lahko uporablja oba). Prenos je realiziran s posebno enoto **DMA krmilnikom**, ki je sposoben sam komunicirati z glavnim pomnilnikom. Posebna izvedba te vrste prenosa je z **vhodno/izhodnimi procesorji**, ki pa je še veliko dražja. Služijo tudi pri

brisanju ali branju v registre V/I naprav, ki so v posebnem naslovnem prostoru. CPE sporoča svoje zahteve V/I procesorjem, ostalo pa opravijo sami. To rešitev uporabljajo samo večji računalniki.

3) Pomnilniško preslikan V/I

Memmory Mapped I/O – registri krmilnikov (glej 2) so kar v pomnilniškem naslovnem prostoru in so iz CPE videti kot pomnilniške besede. Za branje/pisanje lahko uporabljamo kar navadne ukaze za dostop za pomnilnika, tako da posebni V/I ukazi niso potrebni.

4) Lokalnost pomnilniških dostopov

Programi več kot enkrat uporabijo iste ukaze in operande in tiste, ki so v pomnilniku blizu trenutno uporabljenim. Omogoča pomnilniško hierarhijo. Povzroča jo način pisanja programov in že samo delovanje Von Neumannovih računalnikov (ukazi si zaporedno sledijo). Razdelimo jo lahko na:

- **prostorsko lokalnost** – po pojavu naslova bo njegov naslednji po lokaciji v pomnilniku blizu prejšnjega … če ni skoka se ukazi jemljejo zaporedno, v programih nastopajo strukture kot so polja, program je razdeljen na podprograme in procedure, ki so večinoma kratke.
- **časovna lokalnost** – program ob času t tvori naslove, ki jih je tvoril malo pred t in ki jih bo nekoliko po t … zanke (isto zaporedje ukazov se ponovi velikokrat) in začasne spremenljivke, zaradi katerih se naslovi določenih operandov pojavljajo večkrat.

Lahko razložimo še z delovno množico (Working Set) – V(t,T): je veliko manjša od N (število ukazov), vsebina zaporedno si sledečih množic se spreminja počasi – prekrivanje.

5) Amdahlov zakon

Pove nam, kako močno lahko pohitrimo računalnik s paralelizmom določenih delov.
Če za faktor N pohitrimo delovanje pri vseh operacijah, razen pri f-tem deležu od vseh – delovanje je N-krat hitrejše (1-f)-ti del časa, potem je povečanje hitrosti računalnika enaka:

$$S(N) = 1 / (f + (1-f) / N) = N / (1 + (N-1)f)$$

Torej: če hočemo doseči pohitritev, je boljše da imamo en hiter procesor, kot pa več počasnejših, razen seveda v posebnih primerih, ko je stopnja paralelnosti (1-f) visoka.

Skupaj s sodelavcem **R.P. Case-om**, je Amdahl razvil še dve pravili:

- velikost glavnega pomnilnika v bajtih mora biti najmanj enaka številu ukazov, ki jih v sekundi izvede CPE
- zmogljivost V/I sistema v bitih na sekundo mora biti najmanj enaka številu ukazov, ki jih v sekundi izvede CPE

Če računalnik ustreza tem pravilom, rečemo da je **uravnotežen** (izkoriščen).

6) Numerični operandi v fiksni vejici

Vejica je fiksna. Številke na levi strani predstavljajo celo število, na desni pa ulomek. Vsaka številka ima svojo težo glede na pozicijo – pozicijska notacija. Najenostavneje je, da damo vejico čisto na desno in imamo samo cela števila. V glavnem se uporablja dvojiška predstavitev (desetiška se uporablja izključno za poslovne obdelave – banke).

Predstavitev predznačenih števil:

- **predznak in velikost**: najbolj levi bit predstavlja predznak 0 pozitivno in 1 negativno. Pri seštevanju in odštevanju je treba predznak obravavati drugače kot ostale bite. Imamo dve ničli za kateri velja +0>-0. $V(b) = (-1)^{b_{(n-1)}} \sum b_i 2^i$ (suma gre od 0 do n-2)
- **predstavitev z odmikom**: vsa števila so pozitivna, če so vsi biti 0 je to najbolj negativno število, odmik je treba zmeraj popravljati (pri seštevanju odštevati in pri odštevanju prištevati. $V(b)= \sum b_i 2^i - 2^{(n-1)}$ (suma gre od 0 do n-1)
- **eniški komplement**: predstavitev negativnega števila dobimo tako, da najprej zapišemo pozitivnega, zatem pa vse bite invertiramo. Invertiranje bitov je ekvivalentno odštevanju števila od $2^n - 1$. $V(b)= \sum b_i 2^i - b_{n-1}(2^n - 1)$ Prednost je v tem, da vse bite obravnavamo enako. Pri prenosu iz n-1. mesta moramo k rezultatu prišteti 1. Imamo dve ničli.
- **dvojiški komplement**: kot eniški, le da pri invertiranju prištejemo ena. Pri prenosu ni treba prištevati enke. Imamo samo eno ničlo. Najbolj pogosto uporabljen.

Do **prenosa** pride pri prekoračitvi obsega nepredznačenih števil: $x<0$ ali $x>2^n - 1$

Do **preliva** pride pri prekoračitvi obsega predznačenih števil: $x<-2^{(n-1)}$ ali $x>2^{(n-1)}-1$

7) Numerični operandi v plavajoči vejici

Predstavimo lahko tudi ulomke. Število razbijemo na tri dele: mantiso m, eksponent e in bazo r. Eksponent pove na katerem mestu je decimalna vejica, ki plava. Ta števila so predstavljena s fiksno vejico. Zaradi praktičnih razlogov za bazo uporabljamo samo 2 ali 10. Ker je baza konstantna lahko rečemo, da je število v plavajoči vejici predstavljeno s parom (m,e).

Podliv: števila, ki so premajhna za predstavitev, tudi če jih denormaliziramo.

Preliv: glej 6.

8) IEEE 754

- baza = 2
- eksponent je predstavljen z odmikom … če je 0 je število najbližje 0
- mantisa je predstavljena s predznakom in velikostjo
- uporablja implicitno predstavitev normalnega bita
- več formatov: 32 bitni(enojna nat.), 64 bitni (dvojna) in 80 bitni (samo za interno računanje)
- obstaja predstavitev za ∞, -∞ in NaN
- dovoljeno je računanje z nenormaliziranimi števili (ena vrednost eksponenta je rezervirana)

Zaokroževanje:

- zaokrožuje se od matematične natančne vrednosti
- zaokroži se k najbližjemu še predstavljivemu številu
- če je enako oddaljeno od obeh se zaokroži na sodo število

Realizacija: imamo 3 bite

- zaokroževalni bit ima vrednost prvega bita ob najmanjšem, ki pade ven
- varovalni ima vrednost drugega
- lepljivi ima vrednost ena, če je (čisto) ven padla vsaj ena 1

Zaokroževalni	Varovalni	Lepljivi	Število +
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	odvisno
1	0	1	1
1	1	0	1
1	1	1	1

9) Enojna, dvojna natančnost

Enojna:

- predznak S
- 8-bitni eksponent E z odmikom 127
- 23-bitna mantisa m
- vrednost: **(-1)^S (1,m)2^(E-127)**

Dvojna:

- predznak S
- 11 bitni eksponent E z odmikom 1023
- 52-bitna mantisa m
- vrednost: **(-1)^S (1,m)2^(E-1023)**

10) Načini shranjevanja operandov v CPE

Ni nujno, da obstaja možnost shranjevanja operandov v CPE. Kljub temu imajo vsi današnji računalniki v CPE majhen pomnilnik, v katerega je možno shraniti enega ali več operandov – programsko dostopni registri. Omogoča večjo hitrost in krajše ukaze. Razlikujemo tri možnosti za shranjevanje operandov v CPE:

- **Akumulator**: imamo en sam (ali mogoče dva) register, v katerega lahko shranimo en operand. Preprostost, staro. V ukazih ga ni treba eksplicitno navajati. Vmesne rezultate je treba prenašati nazaj v pomnilnik. Posledica: počasnost, promet med CPE in gl. pom.
- **Sklad**: pomnilnik je podoben akumulatorskemu, saj je pri obeh direktno dostopen samo en operand, a vendar je promet med CPE in gl.pom manjši. Sklad je ponavadi narejen tako, da se nadaljuje v glavnem pomnilniku. Take računalnike imenujemo skladovne. Njihova prednost je v računanju matematičnih izrazov v postfiksni obliki, slabost pa v tem, da je takih izrazov malo – več je prireditev.
- **Množica registrov**: število je od 8 do 100. Zelo splošna možnost. Ločimo dve rešitvi, glede na svobodo pri uporabi registrov: vsi so ekvivalentni – splošnonamenski registri. Lahko jih pa razdelimo v dve skupini: ena za aritmetično logične operande, ena pa za računanje z naslovi. Druga rešitev je bolj toga, vendar omogoča prihranke pri gradnji CPE.
  Ta rešitev je pomembna za cevovodne računalnike (registri morajo shranjevati vmesne rezultate).

11) Število eksplicitnih operandov

Manjše število pomeni krajše ukaze in tudi manjšo moč ukazov. Večje število zahteva bolj zapleteno CPE. Na število eksplicitnih operandov vpliva tudi tip operacij, ki jih bo računalnik opravljal.
Računalnike z m eksplicitnimi ukazi imenujemo m-operandni ali m-naslovni, kar ne pomeni, da imajo vsi ukazi m operandov, najpomembnejši ukazi (ALE) imajo m operandov. Imamo pet skupin:

- **3+1 - operandni** (glej 12)
- **3 - operandni**: po letu 1980 praktično vsi, ki imajo operande v registrih. OP3 – OP2 operacija OP1
- **2 - operandni**: OP2 – OP1 operacija OP2, preprostost. OP1 in OP2 sta lahko v enem izmed registrov ali v pomnilniku. 1 ½ naslovni

računalniki – en operand v registru en v pomnilniku. Do 1980 najpogostejša vrsta.
- **1 - operandni**: AC – AC op. AC, računalniki, ki imajo v CPE en ali dva akumulatorja.
- **brez-operandni (skladovni)** računalniki: SKLAD(vrh) – SKLAD(vrh) operacija SKLAD(vrh-1). Operandov ni treba eksplicitno navajati.

12) 3+1 - operandni računalnik

Takih računalnikov danes ni več. EDVAC.
OP3 – OP2 operacija OP1
PC – OP4
Ti računalniki niso imeli pomnilnika za shranjevanje operandov v CPE. Četrti operand določa lokacijo naslednjega ukaza. Taki računalniki so uporabljajo pomnilnik s krožnim dostopom (magnetni boben, zakasnilna linija).

13) Lokacija operandov

Problem lokacije se pojavi večinoma samo pri ALE ukazih, saj pri drugih ukazih že sama narava ukaza določa izbiro. Poleg tega je ta problem omejen skoraj samo na 2 in 3-operandne računalnike, vendar je ravno teh največ. Operandi so lahko shranjeni v CPE, v gl. pom ali v V/I krmilnikih. Uporabljata se samo prvi dve možnosti. Ločimo torej tri variante:
- **Registrsko – registrski** računalniki: vsi operandi ALE ukazov so v registrih CPE. Imenujemo jih tudi LOAD/STORE računalniki, ker je treba vsak operand prenesti iz in nazaj v pomnilnik. Imajo kratke, preproste ukaze. Čas izvrševanja ALE ukazov je vedno enak, vnaprej ga lahko določimo (dobro za paralelizem). Slabost je v tem, da je za rešitev istega problema potrebnih več ukazov, kot pri tistih računalnikih, ki imajo ALE ukaze lahko tudi v pomnilniku. To uporabljajo 3-operandni.
- **Registrsko – pomnilniški** računalniki: eden je v registru ali v pomnilniku, drugi pa so vedno registrih. V to množico spadajo računalniki, ki imajo enega ali dva akumulatorja in registrsko-registrskih ukazov nimajo – akumulatorski računalniki. Pri ALE lahko uporabljamo pomnilniške operande, ne da bi jih prej prenesli v registre. Ukazi so daljši, vendar jih je potrebno manj. Časa za izvajanje ne moremo naprej predvideti, saj je odvisen od lokacije operandov. To uporabljajo 2-operandni.
- **Pomnilniško – pomnilniški** računalniki: vsak operand je lahko ali v pomnilniku ali v registru. So najbolj splošni in omogočajo veliko število rešitev pri istem problemu. Lahko delamo tudi brez uporabe registrov, vendar je to slabo. Ukazi so zapleteni. CISC – VAX.

14) Načini naslavljanja

Način naslavljanja je način podajanja operandov v pomnilniku. Lahko jih razdelimo v tri osnovne skupine:

Takojšnje naslavljanje: operand je podan kar z vrednostjo, operand je del ukaza – dodatni dostop do pomnilnika ni potreben. Temu operandu pravim takojšnji ali literal. Običajno se označuje z znakom #. Korist je največja pri primerjavah in v ukazih za prenos podatkov. Če takojšnjega naslavljanja ni, moramo konstante shraniti v glavnem pomnilniku v tabelo literalov.
Neposredno naslavljanje: operand je podan z naslovom (direktno, absolutno naslavljanje). Ker je naslov del ukaza, se ne spreminja in ga zato imenujemo tudi absoluten. Uporablja se samo za spremenljivke na fiksnem naslovu. Zahteva tudi dolge ukaze (če je pomnilnik dolg). Če želimo pomnilniški prostor povečati, ni kompatibilnosti. Naslov je lahko podan tudi samo z delom naslova.
Posredno naslavljanje: naslov pomnilniškega operanda v ukazu je podan preko neke druge vrednosti – ta je lahko v pomnilniku – pomnilniško posredno n. ali v registrih – registrsko posredno naslavljanje. Pri slednjem je to odmik, naslov izračunamo tako, da seštejemo vsebino registra in odmik. Pri pomnilniškem je potreben dodaten dostop do pomnilnika, poleg tega pa imamo v ukazu spet absoluten naslov. Zato se v večini uporablja samo registrsko, ki ga lahko razdelimo v več skupin:
- **bazno naslavljanje**: v ukazu sta podana naslov registra in odmik. Dolžina registra je daljša ali enaka dolžini pomnilniškega naslova. Imenujemo ga bazni register. Dolžina odmika je variabilna. Če je ta dolžina enaka dolžini pomnilniškega naslova, to imenujemo **indeksno naslavljanje**. Običajno se odmik tu podaja z dvema registroma. Spreminjamo bazni register in ne odmik. **Pred-dekrementno naslavljanje** je lahko bazno ali indeksno in avtomatsko zmanjšuje indeks, **po-inkrementno** pa avtomatsko zvečuje indeks. Skupaj lahko tvorita strukturo sklad. **Velikostno indeksno naslavljanje** namesto prištevanja(odštevanja) odmika uporablja množenje z odmikom.
Pozicijsko neodvisno naslavljanje: omogoča neodvisnost programa od pozicije na kateri je, kar pa lahko rešimo tudi v okviru navideznega pomnilnika. Glavno pri tem je, da program ne vsebuje ukazov, ki vsebujejo absolutne naslove. To je pri posrednem in neposrednem pomnilniškem naslavljanju. Spreminjati moramo program. Če program uporablja samo takojšnje in relativno (naslov je določen z vsebino najmanj enega registra) naslavljanje, lahko pozicijsko neodvisnost naredimo tako, da na začetku delovanja spremenimo samo začetno vsebino prvega registra (indeksnega ali baznega). Nekateri računalniki omogočajo **PC-relativno** naslavljanje – bazni register je kar PC.

15) Ortogonalnost ukazov

Ortogonalnost ukazov pomeni medsebojno neodvisnost parametrov, ki jih ukaz vsebuje. Torej: informacija o operandih je neodvisna od informacije o operaciji in informacija o enem operandu je neodvisna od informacije o ostalih operandih – če spremenimo en operand, to ne vpliva na informacijo, ki podaja ostale in tudi ne na operacijsko kodo. Za vsak operand lahko uporabimo vse načine naslavljanja in vse dolžine operandov (pri istem ukazu).

Dobra lastnost je lažje programiranje, po drugi strani pa odstopanje od ortogonalnosti poenostavi zgradbo CPE in jo naredi hitrejšo. Pri RISC računalnikih je ortogonalnost ukazov izgubil svoj pomen.

16) CISC – RISC dilema

Štetje ukazov: vse možne kombinacije operacijskih kod in načinov naslavljanja.
RISC: večina ukazov se mora izvršiti v eni urini periodi
registrsko-registrskki
fiksno ožičena logika
malo ukazov in malo načinov naslavljanja
vsi ukazi so enako dolgi
dobri prevajalniki (spreminjanje vrstnega reda)
Meritve so pokazale, da so pri približno enaki ceni, RISC-i veliko hitrejši od CISC-ov. CPI je pri RISC veliko manjši. Poleg tega je na RISC računalniku realizacija paralelnega procesiranja veliko lažja.

17) IBM 370 ukazi

To so računalniki, ki so se pod oznako IBM System/360 začeli proizvajati leta 1964 in so do danes doživeli več izboljšav.
Serija spada v skupino 2-operandnih registrsko-pomnilniških računalnikov. Uporablja bazno in indeksno naslavljanje, takojšnje(z uporabo baznega). Obstajata dva nivoja priviligiranosti: nadzorni in problemski. Uporabljajo se v multiprogramskem okolju.

Prvotno so bili ukazi razdeljeni na 5 formatov z 8-bitno operacijsko kodo. Sedaj je formatov 14. Sprva so imeli 143 ukazov, kasneje se je povečalo na 208. Ne spada med RISCe pa tudi med ekstremne CISCe ne.

Kanalski ukazi so ukazi za upravljanje V/I procesorjev.

18) Koraki pri izvrševanju ukaza

- Jemanje ukaza iz pomnilnika (fetch), naslov je shranjen v PC
- Izvrševanje v prejšnjem koraku prevzetega ukaza (execution) - izvršilni cikel. Vsebuje dekodiranje ukaza, prenos operandov v CPE, izvedbo z ukazom določene operacije in shranjevanje rezultata. Po izvedbi se PC poveča za dolžino ukaza razen pri skočnih ukazih, s katerimi lahko v PC vpišem poljubni naslov.

Vsakega od zgornjih korakov lahko razdelimo na več elementarnih. Najmanjše trajanje le-teh je ena urina perioda (časovni interval med prehodom iz nizkega v visoko stanje urinega signala in iz visokega spet v nizko). Ti koraki pa so:

- **prevzem ukaza (IF)**: vsebina PC se preko multiplekserja prenese na pomnilniške naslovne signale. Nato se iz pomnilnika prebere operacijska koda ukaza v IR. Če je zadetek v predpomnilniku se prenese v 1 up, sicer je zgrešitvena kazen.
- **dekodiranje ukaza (ID)**: poleg dekodiranja ukaza se v vmesna registra A,B preneseta izvorna registra, ki ju ukaz določa, za 4 se poveča PC.
- **izvrševanje operacije (EX)**: izračuna se dejanski naslov ali pa se izvede ALE operacija. Med registri IR, A in B se določi katera dva bosta šla na vodili S1 in S2. Določi se tudi operacija, ki se bo opravila in rezultat se da na vodilo D in se zapiše v register, ki ga kontrolna enota določi iz operacijske kode.
- **dostop do pomnilnika (MEM)**: ta korak je potreben samo pri ukazih LOAD (MDR – M[MAR]), STORE (M[MAR] – MDR) in TRAP.
- **shranjevanje rezultata**: samo pri ukazih CALL, LOAD, ALE, MOVER in TRAP
  Rd – C

19) Prekinitve

Prekinitev označuje dogodek, ki povzroči, da CPE začasno preneha izvajati tekoči program ter začne izvajati prekinitveno servisni program. Zahteva za prekinitev pride od zunaj. Prekinitve omogočajo, da drugi deli računalnika pridejo do uslug CPE.
Posebne vrste prekinitev so pasti, ki jo zahteva CPE sama ali na zahtevo programerja. Pasti pridejo od znotraj, poleg tega pa so sinhronske na program, prekinitve niso.

CPE odreagira na prekinitveno zahtevo šele potem, ko dokonča izvajanje trenutnega ukaza. Takojšnja reakcija je veliko kompleksnejša za realizacijo in se uporablja le, če drugače ne gre. Programer lahko omogoči/onemogoči nekatere prekinitve – jih maskira. Po vklopu so prekinitve vedno onemogočene.

Zagotoviti je treba tudi nevidnost prekinitev – izvajanje PSP mora biti za uporabnika nevidno. Vse registre, ki se trenutno uporabljajo je treba shraniti (na sklad ali v registre) in jih zatem obnoviti. Med PSP so prekinitve onemogočene, kasneje se morajo omogočiti.

Potrebno je dobiti tudi naslov PSP, prepoznati je treba napravo, ki je poslala prekinitveno zahtevo. Imamo več rešitev:

- programsko izpraševanje (polling) – pregledajo se registri vseh naprav (v nekem vrstnem redu glede na prioriteto) v katerih eden od bitov pove ali je zahtevala prekinitev, Ta rešitev je počasna.
- Lahko se v CPE pošlje informacija, iz katere ta prepozna izvor zahteve. Med najbolj znanimi načini so tako imenovane vektorske prekinitve (glej 20).

Treba je ugotoviti prioriteto (katera od prekinitev se bo izvedla prva). Poleg tega je na nekaterih računalnikih dovoljeno, da prekinitev z višjo prioriteto prekine prekinitev z nižjo – vgnezdene prekinitve.
CPE ima register, s katerim je določen trenutni prioritetni nivo. Odzove se samo na tiste prekinitve, ki imajo višjo prioriteto od lastne. S pomočjo dodatnega elementa – prekinitvenega krmilnika, lahko to vgradimo tudi v CPE, ki ima samo en bit za omogočanje prekinitev.
Če je na en prekinitveni vhod priključenih več naprav, lahko problem prioritet rešimo s pollingom ali pa z marjetično verigo.

Prekinitev je potrebno potrditi, da naprava umakne svojo zahtevo. Lahko se naredi programsko (PSP bere ali piše v nek register krmilnika naprave), ali strojno (CPE s posebnim kontrolnim signalom obvesti napravo, da je upoštevana).

20) Vektorske prekinitve

Vektorske prekinitve so skupno ime za vse načine prepoznavanja prekinjajoče naprave, pri kateri naprava pošlje v CPE informacijo o naslovu njenega PSP. To pošiljanje se naredi v prekinitvenem prevzemnem ciklu, s katerim CPE pove napravam naj pošlje informacijo o izvoru. Gledano iz CPE je to posebna vrsta branja. Od drugih branj ga loči kombinacija kontrolnih signalov.
Pri nekaterih računalnikih je informacija kar naslov, v glavnem pa je samo del naslova PSP. Naslovu, kjer je shranjen naslov PSP pravimo prekinitveni vektor ali vektorski naslov. Uporablja se kot kazalec v tabelo, ki vsebuje naslove PSP. Torej je del naslova številka vektorja. V principu je možno, da ima ista naprava več PSP-jev. Vektorske prekinitve so tipične za večino po letu 1980 razvitih računalnikov.

21) Realizacija prekinitev pri IBM 370

CPE razlikuje štiri skupine prekinitev in dve skupini pasti. CPE ima štiri prekinitvene vhode, preko katerih prihajajo zahteve za naslednje skupine prekinitev: strojna napaka, eksterna prekinitev, V/I prekinitev, Restart. Najvišjo prioriteto ima strojna napaka, najnižjo pa Restart. Restart ni mogoče onemogočiti. Za iste vrste prekinitev se uporablja marjetična veriga. Prepoznavanje je strojno, vendar ne na vektorski način. Potrjevanje je strojno. Prekinitve lahko maskiramo v registru PSW (če je določen bit 1 je določena p. omogočena). Ob prekinitvi se PSW shrani, vanj pa se shrani nova vsebina prenesena iz enega od šestih naslovov (odvisno od vrste prekinitve/pasti). Z

zamenjavo PSW se naredi skok v PSP, hkrati pa se na fiksne naslove shrani opis prekinitve iz katerega je razviden vzrok prekinitve. Iz PSP se nazaj v program vrnemo z ukazom LPSW (load PSW), ki vanj vrne staro vrednost.

Strojna napaka se aktivira, kadar pride pri delovanju računalnika do napak.

Eksterna prekinitev pride iz ure realnega časa, tipke Interrupt in še 6 virov.

V/I prekinitve pridejo iz V/I procesorjev – kanalov, na katere so priključeni krmilniki, na katere so priključene naprave. Razdeljene so v 8 podrazredov, vsak ima v kontrolnem registru svoj bit, s katerim lahko prekinitve maskiramo. Za programerja so vidni samo podkanali. Vsak ima številko podrazreda, s katero vidimo v katerega spada. Lahko jo spreminjamo. PSP mora ugotoviti iz katerega podkanala je prišla IRQ.
IBM 370 nima veliko podpore za vgnezdene prekinitve. Vsi V/I procesorji uporabljajo isti PSP. Vgnezdene prekinitve je možno narediti z ustreznim maskiranjem podkanalov in programsko realiziranim skladom za shranjevanje starega PSW, vendar to ni potrebno, saj vse podrobnosti pri komuniciranju z V/I napravam opravijo V/I procesorji – prekinitve se uporabljajo predvsem za obveščanje programov da je izvajanje V/I operacij končano, oziroma da je prišlo do napake.

Pasti so razdeljene v dve skupini. Supervisor Call za klic sistemskih programov in za preklop med dvema nivojema privilegiranosti in Program Interruptions, ki se sprožijo ob deljenju z nič, nedefiniranim ukazom …

22) Prekinitve pri Cyber

Razviti so bili predvsem za numerične obdelave v okoljih, ki ne zahtevajo hitrega odzivanja na zunanje zahteve. CPU dobiva prekinitvene zahteve iz V/I procesorjev imenovanih PPU – Peripheral Processing Unit. Zaradi posebne realizacije le-teh, ki uporablja strojno časovno dodeljevanje skupne logike, lahko v vsaki urini periodi zahteva prekinitev zgolj en procesor. Vsak od njih lahko prekine CPE, ki nima možnosti za onemogočenje prekinitev. Izvrši ukaz EXN (exchange jump), CPE dobi signal za prekinitev in odreagira:

- dokonča vse ukaze v besedi na katero kaže PC, v eni besedi so lahko do 4 ukazi
- iz 16 60-bitnih besed dolgega področja v gl.pom., ki je določeno za naslovom v registru V/I procesorja se prenese v CPE nova vsebina vseh registrov, istočasno pa se trenutna vsebina shrani na to področje.

Prepoznavanje izvora se doseže avtomatsko. Potrjevanje ni potrebno, ker do prekinitve zagotovo pride. Transparentnost je zagotovljena avtomatsko. Iz PSP se je mogoče vrniti z ukazom XJ ali EXN, ki ga izvede V/I procesor. V CPE se vrnejo stare vrednosti. Bistvo teh IRQ je v preklapljanju enega uporabniškega programa na drugega in ne v servisiranju zahtev V/I naprav. Možne so tudi vgnezdene prekinitve, če le vsak V/I procesor uporablja drugo področje v glavnem pomnilniku.
V/I naprave ne morejo prekinjati V/I procesorjev.

Pasti delujejo podobno le da jih sproži CPE. Naslov pomnilniškega področja za izmenjavo je naslov najnižje lokacije v pomnilniku, ki je dodeljen tekočem programu. Ta naslov je vedno v registru RA v CPE. Past se sproži ob nedovoljenem naslovu, prelivu (za FLOAT), uporabi nedef operanda (NaN). Možno jih je onemogočiti.

23) Vrste nevarnosti v cevovodu

Veliko prednosti cevovodnega procesiranja se zmanjša zaradi cevovodnih nevarnosti, zato jih moramo na nek način odpraviti. Prvi RISC-i so jih odpravljali večinoma programsko (vstavljanje NOP-ov). Tako odpravljanje bistveno poenostavi delovanje CPE, a povzroči nekompatibilnost programov na različnih računalnikih in upočasni delovanje.

Poznamo tri vrste nevarnosti:
Strukturne nevarnosti so kadar dve ali več stopenj cevovoda v isti urini periodi potrebujeta enoto, ki je sposobna delati samo eno operacijo naenkrat (registri, ALE, pomnilnik) ali če hočeta dva ukaza pisati naenkrat v svoj programsko dostopni register. Do teh nevarnostih pride tudi pri ukazih za operacije, ki trajajo več urinih period. Če imamo take operacije v stopnji EX in imamo samo eno tako stopnjo, morajo kasnejši ukazi čakati na to stopnjo. Mogoče jih je odpraviti z zmogljivejšim predpomnilnikom in tako, da vgradimo dovolj veliko število enot (stopnja EX se lahko naredi v obliki cevovoda).
Podatkovne nevarnosti lahko imenujemo tudi operandne nevarnosti. Do njih pride, če nek ukaz potrebuje operand, ki še ni dostopen. Odpravimo jih lahko tako, da se stopnja ID ukaza, ki potrebuje nedostopen operand ustavi, dokler le-ta ni dostopen (cevovodna zaklenitev ≠ cevovodna ustavitev). Vstavljamo mehurčke v vse stopnje, ki so levo od tam, kjer se je cevovod zaklenil. Rešimo jih lahko tudi s premoščanjem, ki je boljša rešitev. Logika za premoščanje omogoča, da dobimo operand iz vmesnih registrov cevovoda, ampak tudi premoščanje ne odpravi vseh nevarnosti, zato je v nekaterih primerih potrebno vstavljanje mehurčkov. Tretji način za odpravljanje podatkovnih nevarnosti, je s spreminjanjem vrstnega reda – cevovodno razvrščanje.
Podatkovne nevarnosti lahko razdelimo v tri vrste: **RAW, WAR, WAW**.
Kontrolne nevarnosti povzročajo v večini bistveno večjo izgubo kot strukturne in podatkovne nevarnosti. Do njih pride pri operacijah, ki spremenijo vsebino PC drugače kot običajno – pri kontrolnih ukazih (pogojni skoki, brezpogojni skoki, klici in vrnitve – SKOKI). Kadar stopnja EX spremeni vsebino PC sta vsebini stopenj IF in ID neveljavni. Najpreprostejši način je, da se namesto vsebine IF in ID vstavita mehurčka. Ta rešitev deluje brez čakanja – dokler cevovod ne ugotovi, da je pogoj za skok izpolnjen, deluje normalno naprej – predpostavlja neizpolnjen pogoj. CPI občutno naraste, zato je treba izgubo zmanjšati. Pri zasnovi cevovoda upoštevamo dve stvari: preverjanje pogoja za skok in izračun skočnega naslova naj se izvaja čim bližje prvi stopnji cevovoda.
Kontrolne nevarnosti lahko odpravljamo tudi s predikcijo. Delimo jo v dve skupini: statično in dinamično. GLEJ NAPREJ

24) Cevovodno procesiranje

Razlog: v zadnjih desetih letih se je najvišja hitrost logičnih elementov povečala za 10X, število elementov na istem čipu pa za 5000X.

Z besedo cevovod se označuje CPE, ki naenkrat izvršuje več ukazov, tako da se posamezni koraki izvrševanja prekrivajo. Podoben je tekočemu traku. Izvrševanja ukaza se razdeli na manjše podoperacije – stopnje, segmente cevovoda. Pomik iz ene v drugo se dela naenkrat. Podoperacije morajo biti uravnotežene. Pospešitev idealnega cevovoda bi bila N-kratna, pri čemer je N stopnja cevovoda, a vendar stopnja ne presega 10, saj z večanjem stopnje cevovodne nevarnosti izničijo pridobitve in je lahko cevovod še slabši kot bi bil, če bi bila stopnja manjša. Cevovod je mogoče narediti na način, ki je za programerja neviden.

25) Podatkovne nevarnosti

Glej 23.

26) Statična predikcija

Pri tej rešitvi sodeluje prevajalnik, ki skuša za pogojne skoke napovedati čim bolj verjeten rezultat pogoja za skok, še preden se program začne izvajati, tako da se napoved med izvajanjem ne spreminja – zato je statična.
Statična predikcija uporablja zakasnjene skoke – poskušamo zapolniti skočne reže. Ne glede na izpolnjenost pogoja se vedno izvršijo vsi prevzeti ukazi. Če hočemo zagotoviti pravilnost, moramo skočni ukaz prestaviti za število skočnih rež ukazov nazaj, v skočne reže moramo vstaviti ukaze, če se da, sicer NOP-e.
Cevovod z zakasnjenimi skoki lahko izboljšamo z uvedbo razveljavitvenih skokov. Običajna napoved je, da skok bo. Če je ta napoved napačna, se ukazi, ki so v skočnih režah razveljavijo. S tem lahko skočne reže skoraj zmeraj zapolnimo.
Zakasnjeni in razveljavitveni skoki občutno prispevajo k zmanjšanju škode. Tudi njihova vključitev je preprosta. Slabost je v tem, da je potrebno drugačno programiranje in da so programi nezdružljivi z drugimi računalniki. Poleg tega je uspešnost statične predikcije pri cevovodih z veliko stopnjo, z velikimi skočnimi režami veliko slabša. Zato se po letu 1990 vse bolj uporablja dinamična predikcija.

27) Dinamična predikcija

Napovedovanje izpolnjenosti pogoja se spreminja med izvajanjem programa. Prilagaja se dogajanju v programu, zato je tudi boljša od statične predikcije.

Najpreprostejša vrsta te predikcije je **prediktorska tabela**. To je majhen pomnilnik, do katerega se dostopa pri pogojnih skočnih ukazih, s spodnjimi biti naslova na katerem je ta ukaz. V najenostavnejši izvedbi so njene vrednosti 1-bitne. Če je bil pogoj izpolnjen se v tabelo vpiše 1, sicer 0. Pri stopnji IF, se bere tudi iz tabele. Če v IF ni pogojnega skoka, se branje razveljavi, sicer se uporabi za napoved skoka. Naslednji ukaz se prevzame iz naslova, ki ga napoveduje bit. Če se v stopnji EX izkaže, da je bila napoved napačna, se stanje bita spremeni, prevzeti ukazi pa se spremenijo v mehurčke. 1-bitna tabela ima slabo natančnost napovedovanja (pri vgnezdenih zankah bo napoved skoraj vedno dvakrat napačna – ob zadnjem (se ne da izogniti) in ob prvem obhodu).Ta pomankljivost se lahko odpravi z 2-bitno tabelo. Če je pogoj izpolnjen se prišteje 1 (če je 3 ostane 3), sicer se odšteje 1 (0-1=0). Če je 2 ali 3 se vzame izpolnjen pogoj, sicer neizpolnjen. Natančnost napovedi je približno 93%. Čeprav je to veliko, se da še izboljšati: upoštevamo tudi preteklost drugih pogojnih skokov in ne samo trenutnega. Prediktorji, ki to upoštevajo se imenujejo **korelacijski prediktorji**. (m,n) korelacijski prediktor uporablja obnašanje zadnjih m skokov, da izbere eno od običajnih prediktorskih tabel velikosti n, v kateri se izbere vrednost na lokaciji, ki jo določimo z zadnjimi biti naslova ukaza. Za vsak skok potrebujemo 2^m tabel. Podatki o zadnjih m skokih se označuje kot globalna zgodovina pogojnih skokov.

Skočni predpomnilnik: poleg izpolnjenosti pogoja mora biti znan tudi skočni naslov. Dobimo ga s pomočjo skočnega predpomnilnika. V njem so shranjeni podatki o zadnjih skokih. Običajno je realiziran skupaj s prediktorskimi tabelami, tako da sta oba del **skočne enote**. Pri zadetku se iz skočnega predpomnilnika naloži vrednost v IR. Če je pogoj izpolnjen se ta vrednost prenese iz IR v PC. Ko pride ukaz v EX se preveri izpolnjenost pogoja za skok in enakost napovedanega skočnega naslova – zaradi t.i. indirektnih skokov, pri katerih je skočni naslov določen z vsebino določenih registrov, ki pa se lahko med izvajanjem spreminjajo. Lahko imamo proceduro, ki se kliče iz več mest. To lahko rešimo s skladom.
Če je skočni pogoj ob preverjanju neizpolnjen, se ukaz izbriše iz skočnega predpomnilnika. Skočnemu ukazu, ki je sedaj v stopnji EX, se dovoli, da v PC v stopnji IF prenese izračunani skočni naslov (PC-PC2+4).
Če je skočni pogoj izpolnjen, napovedani naslov pa ni enak izračunanemu, se izračunani zapiše v skočni predpomnilnik. Skočni ukaz v PC prenese izračunani skočni naslov: PC-PC2+4+razR.

28) Dinamična predikcija in skočni predpomnilnik

Glej 27.

29) Kako doseči CPI<1

Cevovod omogoči da se CPI približa 1. Če želimo CPI zmanjšati pod 1, moramo v eni urini periodi prevzeti več kot 1 ukaz – več izstavitveni

procesorji. Te procesorje lahko razdelimo v superskalarne in VLIW procesorje. Razlika je v načinu prevzemanja in izstavljanja ukazov. Prevzemanje ukazov in njihovo izstavljanje sta operaciji, ki ju je mogoče izvajati statično ali dinamično.

Statično razvrščanje: procesor prevzema ukaze v natanko takem vrstnem redu, kot so v programu – inorder. Spreminjanje vrstnega reda lahko opravlja prevajalnik. Ni potrebno imeti logike, ki spreminja vrstni red prevzemanja. Slabost je, da ne pridejo v istočasno izvrševanje ukazi, za katere se med izvajanjem programa pokaže da bi lahko prišli, vendar prevajalnik tega ne more ugotoviti in zato ne razvrsti ukazov.

Dinamično razvrščanje: procesor prevzema ukaze po vrstnem redu, ki je drugačen od tistega v programu – out of order. Spreminjanje vrstnega reda opravlja procesor. Kadar je ena od enot procesorja prosta išče ukaze, ki jih je mogoče poslati vanjo. Prednost se pokaže pri zgrešitvah v predpomnilniku, med čakanjem lahko izvrši nek drug ukaz. To je mogoče narediti samo pri neblokirajočem pomnilniku. Pri iskanju ukazov moramo uporabljati dinamično predikcijo skokov. Ker predikcija ni vedno pravilna, ni zanesljivo, da bo tako prevzet ukaz v resnici potreben, zato se tako izvrševanje imenuje špekulativno.

Statično izstavljanje: je način pri katerem je vrstni red izstavljanja ukazov v stopnjo EX določen že pri prevzemu ukaza. Procesor glede tega ne odloča o ničemer. Taki procesorji imajo običajno tudi statično razvrščanje.

Dinamično izstavljanje: vrstni red izstavljanja določa logika v procesorju. Pregleduje ukaze, ki se izvršujejo in išče take, ki niso odvisni od trenutno izvršujočih. Če takega najde, ga izstavi v izvrševanje. Pri špekulativnem izvrševanju se rezultati ukazov ne smejo shraniti v programsko dostopne registre ali pomnilnik tako dolgo, dokler ni gotovo, da je bila predikcija skokov, s katero so bili prevzeti pravilna.

Pri **superskalarnih** procesorjih je izstavljanje ukazov vedno dinamično. Po letu 1995 je začelo prevladovati tudi dinamično razvrščanje. Videti je kot da imamo več cevovodov.

Pri **VLIW** procesorjih ne prevzemamo enega ampak več ukazov naenkrat. Vsak ukaz ima svojo cevovodno enoto. Uporablja statično razvrščanje in izstavljanje. Težko je najti probleme, ki imajo dovolj veliko količino paralelnosti.

30) Načini dostopa do glavnega pomnilnika

Poznamo dva načina glede na način dostopanja do pomnilniški besed: s podajanjem naslova in s podajanjem vsebine besede (asociativni predpomnilniki). Navadne pomnilnike delimo še glede na vrstni red naslovov do katerih dostopa:
Naključni dostop: čas dostopa do besede je neodvisen od naslova pred tem naslovljenih besed – je konstanten. To so DRAM pomnilniki, ki imajo tudi

page mode način, ki pa ni naključni; dostopamo do zaporednih bitov, ki so v prebrani vrstici. Ta način pokaže svoje prednosti pri uporabi predpomnilnika, saj se vanj vedno prenese cela vrstica (ena ali več).
Zaporedni dostop: čas za dostop je odvisen od naslova besede, do katerega smo dostopali prej. Izvršiti moramo dostop do vseh vmesnih besed. Magnetni trak, pomikalni reguister.
Krožni dostop: posebna vrsta zaporednega dostopa – magnetni diski s fiksnimi glavami. Predstavljamo si ga lahko kot zlepljen magnetni trak.
Direktni dostop: kombinacija krožnega in zaporednega dostopa (najprej se glava premakne na pravo sled (zaporedni), potem pa se sled vrti, dokler glava ne doseže želenega naslova (krožni)) – magnetni disk.

31) Organizacija glavnega pomnilnika

Vidik CPE nad pomnilnikom imenujemo organizacija pomnilnika.
Glavni pomnilnik je videti kot enodimenzionalno zaporedje pomnilniških besed.
Pomnilniška beseda je najmanjše število bitov s svojim naslovom.
Pomnilniški naslov je število, ki enolično določa neko pomnilniško besedo. Z n bitnim naslovom lahko naslovimo 2^n besed – dolžina določa maksimalno velikost pomnilniškega prostora.

Ločimo signale preko katerih se naslavlja (naslovni) in signale preko katerih se prenašajo (podatkovni) iz/v pomnilnik.

Dolžina besede določa tudi dolžino registrov (mnogokratnik dolžine besede) - posredno. Največji izkoristek je, če je dolžina besede 1, saj to povzroči veliko bolj zapleteno in počasno CPE.

Obstaja ena sama napaka, ki jo je pri kasnejšem razvoju računalnika težko popraviti – premajhno število bitov za pomnilniški naslov.

Iz pomnilnika lahko prenašamo več besed naenkrat – odvisno od širine poti.

Problem poravnanosti se pojavi, če je širina poti daljša od ene besede in če je operand, ki ga želimo prebrati, na naslovu ki ni deljiv s širino poti. Potem moramo opraviti dve branji in ju združiti v en operand.
Če dovolimo neporavnanost, je prostor bolj izkoriščen, porabi pa se več časa in pot med CPE in pomnilnikom je bolj zasedena.

32) Metabiti (označevalni biti)

To so biti pomnilniške besede, ki opisujejo pomen ostalih bitov. Večina računalnikov jih nima ali pa uporablja samo bite za detekcijo/korekcijo napak – paritetne bite (SECDED). Vendar pri pojmu metabiti mislimo na bite, ki povejo kaj dana beseda vsebuje. Z njimi dosežemo to, da se CPE začne zavedati zgradbe programa, kar povzroči:

- manjše število ukazov (če ni ortogonalnosti)
- avtomatska pretvorba operandov
- avtomatsko računanje in preverjanje indeksov
- avtomatsko vzpostavljanje klicnih parametrov
- ugotavljanje nesmiselnih operacij
- ugotavljanje nedefiniranih operandov

Glavna prednost je lažje programiranje v zbirnem jeziku. Semantični prepad je manjši. Ker se na tem področju ne programira veliko, je upadlo tudi zanimanje za metabite. Slabost je tudi v bolj kompleksni CPE, kar povzroči težjo realizacijo paralelnosti.

33) Zaščita glavnega pomnilnika

Nevarnosti: programer sesuje operacijski sistem s spreminjanjem le-tega, programi se lahko tepejo v multiprogramskih sistemih…Prvi računalniki niso imeli nobene zaščite, saj so se uporabljalo na enouporabniški način. Za zagonske in nekatere druge programe so uporabljali bralni pomnilnik.

Najpreprostejši pravi način za zaščito je par registrov, ki vsebuje spodnjo in zgornjo mejo naslova, ki pripada programu. Lahko imamo tudi samo začetni naslov in dovoljeno dolžino programa. Slabost je v tem, da mora program biti zvezen, poleg tega pa so vse besede zaščitene na enak način (IBM 1800 – metabiti).

Rešitev, ki se uporablja danes je taka: glavni pomnilnik je razdeljen na dele določene velikosti, ki so vsak zase zaščiteni. Tem delom pravimo bloki ali strani. Vsakemu programu je dodeljen pomnilniški prostor, ki je enak celemu številu strani. Vsaka stran ima svoje zaščitne metabite – zaščitni ključ, ki se shrani v poseben pomnilnik, do katerega ne moremo dostopati s pomnilniškimi ukazi.

34) Zaščita glavnega pomnilnika pri IBM 370

Stran je velika 4096 besed. Zaščitni ključ je dolg 7 bitov na stran. Bit 4 določa način delovanja zaščite za stran. Če je 0 deluje samo pri pisanju, sicer tudi pri branju. Pri vsakem dostopu, se biti 0-3 ključa primerjajo z biti 8-11 v PSW, oziroma če dostopa V/I naprava z biti 0-3 v drugi besedi ORB bloka. Dostop je dovoljen, samo če so ti biti enaki.
Zaščitni ključi so v posebnem pomnilniku in jih lahko spreminjamo z SET STORAGE KEY in berem z INSERT STORAGE KEY – privilegirana ukaza. Programi, ki delujejo v nadzornem načinu delovanja, lahko spreminjajo ključe in obidejo zaščito, zato je treba zagotoviti, da navaden uporabnik ne more delovati v nadzornem načinu (SUPERVISOR CALL).

35) Predpomnilnik

Predpomnilnik izrablja lokalnost pomnilniških dostopov: je majhen in hiter. Na zunaj deluje kot da je velik. Zgrajen je iz SRAMov in je običajno na istem vezju kot CPE. Služi kot premoščanje vrzeli med hitrostjo DRAMov in hitrostjo CPE. Vrzel se zmanjša iz 750X na 15X.
Da je učinkovitejši, ga razdelimo na dva dela (heterogen - Harvardski predpomnilnik) – enega za operande in enega za ukaze, tako da se lahko v isti urini periodi dostopa do obeh. Uporabimo tudi širše podatkovne poti iz CPE do predpomnilnika in iz predpomnilnika do glavnega pomnilnika. Dodamo lahko tudi drugi, tretji, … nivo predpomnilnika.

Predpomnilnik imamo lahko tudi v V/I procesorjih ali napravah.
Je podmnožica glavnega pomnilnika.

Predpomnilnik tipično predstavlja 1% glavnega. Če je zgrešitev, je treba podatke prenesti iz glavnega oziroma iz L2. Predpomnilnik razdelimo na bloke, ki so veliko toliko, kot je široka prenosna pot. Če je zgrešitev, se prenese cel blok.

Vsak blok ima podatkovni del in kontrolni del, v katerem so naslov bloka, umazani in veljavni bit. Iz naslova bloka lahko razberemo še naslov posamezne besede v njem. Če je bit V enak 0 je vsebina bloka neveljavna (ob zagonu računalnika). Dostop do takega bloka je zgrešitev. Če je prišlo do pisanja v blok se U postavi na 1.

36) Predpomnilniki glede na omejitve pri preslikavi

NASLOV:

n-b bitov – naslov bloka	b bitov – naslov besede v bloku

Problem je v primerjavi naslova, ki ga da CPE z zgornjimi n-b biti v kontrolnem delu bloka. Primerjavo je treba opraviti z vsemi bloki v eni urini periodi. Zato uvedemo določene omejitve pri preslikavi.

Problem primerjave lahko rešimo z uporabo asociativnega pomnilnika. Pri predpomnilniku predstavljajo vsebino naslovi, ki so shranjeni v kontrolnem delu, tako da kontrolni del realiziramo asociativno. CPE takoj ugotovi ali imamo zadetek ali zgrešitev. Takemu predpomnilniku pravimo **čisti asociativni** in nima nobenih omejitev pri preslikovanju. Pomnilnik je popolnoma izkoriščen. Problem pa je v tem, da zadosti velikih asociativnih predpomnilnikov z današnjo tehnologijo ni mogoče narediti (dosegajo velikosti do nekaj 100 besed).

Velik predpomnilnik lahko naredimo le z vpeljavo omejitev pri preslikovanj naslovov. Namesto enega velikega asociativnega pomnilnika ga razbijemo na več manjših. Predpomnilnik razbijemo na več setov (2^S) – dobimo **set-asociativni predpomnilnik**. Število blokov v setu imenujemo stopnjo

asociativnosti (2^E) in je tipična med 2 in 8. Sedaj je za vsako besedo glavnega pomnilnika vnaprej določeno v kateri set se bo preslikala.
Za nek naslov Ai velja da se lahko preslika v enega izmed blokov seta, ki ga določa enačba Si=Ai(b (mod2^s) n-1)

Če je stopnja asociativnosti enaka 1, dobimo t.i. **direktni predpomnilnik**, pri kateremu so omejitve pri preslikavi najhujše. Za vsako besedo je vnaprej določeno v katerega izmed blokov(=setov) se bo preslikala.

Z zmanjševanjem stopnje asociativnosti se zmanjšuje tudi cena in verjetnost zadetka.

Četrta možnost je **pseudo asociativni predpomnilnik**. Poskuša združiti prednosti direktnega in set-asociativnega pomnilnika. Če imamo zadetek deluje identično kot direktni predpomnilnik. Sicer se namesto dostopa do višjega nivoja dela dostop do pseudo bloka. Dostop do tega bloka je nekoliko daljši. Obstaja nevarnost, da so vsi zadetki v pseudo bloku. Če se zgodi to, preprosto zamenjamo vsebino blokov.

37) Predpomnilniško pravilo 2:1

V zvezi z direktnim predpomnilnikom velja naslednje pravilo: verjetnost zgrešitve direktnega predpomnilnika velikosti M je približno enaka verjetnosti zgrešitve set asociativnega predpomnilnika s stopnjo asociativnosti 2 in velikosti M/2.

38) Kakšna naj bo velikost bloka

Predpomnilnik lahko povečujemo tako, da povečujemo stopnjo asociativnosti, število setov ali velikost bloka(najlažje in najceneje).
S povečevanjem bloka se ustvarjajo pogoji za boljšo izkoriščenost prostorske lokalnosti, ker pa se s povečevanjem pri isti velikosti predpomnilnika zmanjšuje število blokov, se s tem slabšajo pogoji za izkoriščanje časovne lokalnosti. Iz meritev lahko ugotovimo, da so pri večjih predpomnilnikih boljši večji bloki in obratno za manjše. Če je velikost blokov premajhna, se le-ti stalno zamenjujejo in verjetnost zadetka je majhna.

39) Kateri blok naj se zamenja ob zgrešitvi

Pri direktnih predpomnilnikih je ta izbira preprosta, saj imamo samo eno možnost.
Drugače je pri (set) asociativnih predpomnilnikih. Če je blok ki ga mečemo iz predpomnilnika umazan, ga moramo najprej prenesti nazaj v glavni pomnilnik.
Za določanje blokov za prenos se uporabljata dve strategiji:

- Naključna strategija: blok se izbere naključno

- LRU: Least Recently Used strategija beleži uporabo blokov in zamenja tistega, ki se je uporabljal najmanj. Izkorišča časovno lokalnost.

Prednost naključne strategije je v enostavnosti logike, pri LRU za asociativno ≥4 postane logika težka za realizacijo. Lahko se naredi samo na približno. Pri večjih stopnjah asociativnosti se uporablja naključna strategija. Če je pomnilnik velik in je tudi asociativnost velika, med strategijama sploh ni več razlike.

40) Pisanje v predpomnilnik

Ker je veliko več dostopov do pomnilnika bralnih, je dobro da ga optimiziramo za branje. A vendar moramo zaradi Amdahlovega zakona realizirati tudi hitro pisanje. Imamo dve strategiji pisanja:

- **pisanje skozi**: vedno se piše tako v predpomnilnik kot tudi v glavnega.
- **pisanje nazaj**: piše se samo v predpomnilnik. Ko pride do zgrešitve in se blok, v katerega smo prej pisali zamenja, se mora ta blok zapisati nazaj v glavnega. Ali ga je treba prenesti ali ne označimo z umazanim bitom.

Prednost pisanja nazaj je v tem, da pisanje poteka s hitrostjo, ki jo ima predpomnilnik. Pri več pisanjih v isti blok je potrebno samo eno pisanje v glavni pomnilnik. Ob prenosu bloka nazaj se lahko izkoristi maksimalna hitrost prenosa.
Prednost pisanja skozi je enostavnost in vsebinska skladnost. Potrebujemo pisalni izravnalnik, ki do neke mere razbremeni CPE. Sicer ima prostor za več pisanj (8-16), ampak se lahko napolni, tako da mora CPE vseeno čakati. Pisalni izravnalnik je realiziran kot čisti asociativni pomnilnik.

Pisanje v 1 urini periodi realiziramo cevovodno. Ob zadetku se podatek skupaj z naslovom zapiše v shranjevalni register. Če ni zadetka, se ob koncu periode vsebina registra razveljavi.
Ob naslednjem pisanju se veljaven podatek iz shranjevalnega registra zapiše na naslov, ki je v shranjevalnem registru. Istočasno se vanj zapiše že nov podatek z naslovom.

Pri pisalnih zgrešitvah se uporabljata dve rešitvi:

- **pisalna zamenjava**: pisalne zgrešitve se obravnavajo enako kot bralne. V predpomnilnik se prenese nov blok, ki mu sledi dostop do predpomnilnika. Ta način se uporablja pri obeh strategijah pisanja.
- **pisanje naokrog**: blok se spremeni samo v glavnem pomnilniku

41) Vrste zgrešitev pri predpomnilniku

Obvezne (mrzle) zgrešitve: ob prvem dostopu do neke besede. Če bi imeli večje bloke, bi bilo teh zgrešitev manj, vendar s tem povečamo zgrešitveno kazen – ni vredno.
Velikostne zgrešitve: ker je predpomnilnik končne velikosti, slej ko prej v njem zmanjka prostora – ne more vsebovati vseh blokov, ki jih med izvajanjem potrebuje. Edina rešitev je, da predpomnilnik zvečamo.
Konfliktne zgrešitve: pojavijo se samo pri set asociativnih in direktnih predpomnilnikih, ko je v predpomnilniku sicer še dovolj prostora, a ima blok določeno preslikavo na že zaseden set. Teh zgrešitev ne bi bilo, če bi bil predpomnilnik čisti asociativen. Zmanjšamo jih lahko s povečanjem stopnje asociativnosti.

42) Problem skladnosti v predpomnilniku

Skladnost označuje pojav, da je vsebina nekega bloka v predpomnilniku enaka vsebini istega bloka v glavnem pomnilniku. Ta problem se pojavlja predvsem pri pisanju nazaj, v neki meri pa tudi pri pisanju skozi.
Neskladnost povzroča napačno delovanje zaradi prenosov med V/I napravami in glavnim pomnilnikom in zaradi shranjevanja podatkov iz glavnega pomnilnika na disk.
Problem neskladnosti, ki ga povzroča V/I lahko rešimo tako:

- priključimo jih tako da grejo skozi predpomnilnik: zapletena rešitev, verjetnost zadetka se občutno poslabša – počasnejše delovanje rač.
- pred izvrševanjem V/I prenosov se razveljavi vsebina predpomnilnika ali pa se vsi umazani bloki prenesejo. To je programski način in ne zahteva dodatne logike. To deluje, ker so V/I prenosi redki.
- z dodatno logiko naredimo mehanizem, ki selektivno razveljavlja ali prazni predpomnilnik – za računalnike, ki imajo V/I procesorje ali več CPE. Nadzor nad prenosi ni več pod kontrolo CPE zato programska rešitev ni možna. Imamo pa dva mehanizma:

1. **centralni imenik**: informacija o naslovih vseh blokov, ki so trenutno v enem od predpomnilnikov je shranjena v centralnem imeniku, ki je vgrajen v krmilnik pomnilnika. Če pride do pisanja v blok, se to registrira v imeniku. Če je ta blok še v drugih predpomnilnikih, krmilnik poskrbi, da se zamenja/razveljavi.
2. **vohunjenje (snooping)**: uporablja se kadar so procesorji in glavni pomnilnik priključeni na isto vodilo, po katerem grejo vsi dostopi do njega. Vsi procesorji znajo vohuniti na vodilu. Procesor, ki spremeni besedo v svojem predpomnilniku pošlje na vodilo njen naslov skupaj z vsebino in signalom. Tako vsi ostali procesorji ugotovijo, če je bila spremenjena beseda, ki se nahaja v njihovem predpomnilniku. Tako jo spremenijo ali pa razveljavijo. Če je bilo samo branje in je bila prebrana ista spremenjena beseda, potem lahko procesor obvesti procesor ki bere, naj opusti branje, zapiše spremenjeni blok v glavni pomnilnik in ga obvesti da naj ponovi branje.

S problemom skladnosti se ukvarjajo tudi **skladnostni protokoli**, ki določajo kaj naj se zgodi z bloki v predpomnilniku ob dostopu do njih. Med njimi je danes najbolj znan **MESI** (Modified, Exclusice, Shared, Invalid). Njegovo ime je okrajšava za 4 stanja bloka, ki jih predvideva. Zasnovan je na osnovi vohunjenja. Za programerja je neviden in deluje avtomatsko. Uporablja se v L2 in v operandnem predpomnilniku.

43) Zmanjševanje zgrešitvene kazni

Zmanjšamo jo lahko s pametnim vrstnim redom prenašanja besed v bloku. Drugače imamo še dve zelo pomembni rešitvi:

- **vnaprejšnji prevzem bloka**: velika verjetnost je da bo poleg bloka, ki ga prevzamemo potreben tudi višji. Prebere se še višji blok in se shrani v bralni izravnalnik. To je učinkovito, le če je prenos dveh blokov približno enako hiter kot prenos enega.
- **neblokirajoči predpomnilnik**: ob zgrešitvi se CPE ne ustavi, ampak vnaprej ali špekulativno izvršuje druge ukaze. Potrebujemo neblokirajoči pomnillnik, način delovanja imenujemo zadetek pod zgrešitvijo. Smiseln je samo, če povezava med CPE in gl. pom. omogoča, da se prenaša več blokov naenkrat.

44) Pomnilniško prepletanje

Zmanjšanje dostopov do glavnega pomnilnika ima spodnjo mejo, saj je enkrat treba vrniti rezultate nazaj v glavni pomnilnik oziroma jih prebrati iz njega. Page mode način do neke mere pomaga. Preostane pa nam še to, da povečamo število naenkrat prenesenih besed. To lahko naredimo tako da:

- razširimo podatkovne poti do glavnega pomnilnika – **širina pomnilnika**
- s **pomnilniškim prepletanjem**: glavni pomnilnik razdelimo na m samostojnih delov – modulov. To pomeni da lahko poteka največ m dostopov istočasno.

Veliki računalniki uporabljajo dostop do modulov po širokih podatkovnih poteh.
Pomnilniško prepletanje je koristno tudi, ker omogoča istočasne dostope do glavnega pomnilnika na računalnikih, ki imajo več CPE / V/I procesorje. Pomembno nalogo ima tukaj **krmilnik pomnilnika**, ki iz pomnilniških naslovov, ki jih dajo procesorji ugotovi na kateri modul se nanašajo. Dostop se opravi takoj, ko postane modul prost. Če želi dostopati prej, imenujemo to konfliktni dostop.

Pomembno je, da zagotovimo da so podatki po modulih čimbolj enakomerno razporejeni.
Najpogostejše prepletanje je spodnje prepletanje, pri katerem je modul določen s spodnjimi biti pomnilniškega naslova. To prepletanje omogoča

izkoristiti zaporedno lokalnost. Verjetnost, da se bodo zaporedni naslovi nanašali na različne module je precejšnja. Pri takem prepletanju in pri m modulih je v povprečju možno √m istočasnih dostopov. Temu ustrezno se poveča tudi hitrost prenosa informacije v/iz glavnega pomnilnika.

45) Vrste navideznega pomnilnika

Najstarejša vrsta navideznega pomnilnika je **ostranjevanje** – Atlas. Pomožni pomnilnik je razdeljen na bloke s fiksno dolžino – strani. Vse strani skupaj sestavljajo pomožni pomnilnik. Podobno je razdeljen tudi glavni pomnilnik – na okvire strani. Vsako stran je mogoče prenesti v poljuben okvir. Tipične velikosti strani danes so od 4KB do 16KB. Vsak program zaseda določeno število strani. Ker je velikost programa redko večkratnik velikosti strani, je v povprečju neizkoriščene pol strani na program – NOTRANJA FRAGMENTACIJA. Preslikovalna funkcija je pri ostranjevanju definirana s pomočjo posebne tabele – tabele strani. Vsaki strani pripada ena polje – deskriptor strani. Fizični naslov določimo s pomočjo navideznega tako, da ga razbijemo na dva dela. Spodnji del (p bitov) določa naslov besede znotraj strani. Zgornjih n-p bitov določa številko strani. Do deskriptorja pridemo tako, da seštejemo številko strani z registrom tabele strani, ki vsebuje njen naslov. Informacijo v deskriptorju sestavljajo:

- Veljavni bit V: če je 1, so parametri v deksriptorju veljavni
- Prisotni bit P: če je 1, je stran v enem od okvirov v glavnem pomnilniku – stran je aktivna. Imamo zadetek, sicer imamo zgrešitev – napako strani.
- Zaščitni ključ RWX sestavlja več bitov, ki povejo kakšna vrsta dostopa je dovoljena in od koga.
- Umazani bit C je vedno 0, ko se stran prenese v okvir. Če pride med izvajanjem do spremembe v strani se postavi na 1.
- Številka okvira FN podaja številko okvira, v katerem je stran. Veljavna je samo, če je P=1. Pri dovoljenem dostopu se sešteje FN X 2^p + naslov znotraj strani, ki jih določa zadnjih p bitov navideznega naslova.

Tako preslikovanje imenujemo linearno – naslovi se širijo brez omejitev pri prehodu iz enega na drugega. Naslovni prostor je raven (flat). Delitev na strani je za uporabnike nevidna.

Dobra lastnost ostranjevanja je njegova preprostost, a ne uporablja zelo dobrih načinov za zaščito. Več programov si ne more deliti isti prostor. Vsebina blokov nima svojega pomena – delitev na bloke je povsem mehanična.

Pri navideznih pomnilnikih, ki upoštevajo zgradbo programov je to drugače. Imenujemo jih **segmentacija**. Prva razlika je v različni dolžini segmentov. Dolžina ni fiksna, poleg tega pa ima vsak segment svoj pomen (strojni modul – preveden programski modul: procedura, tabela…). Med izvajanjem se prenese v glavni pomnilnik na poljuben naslov.

Beseda je določena glede na naslov segmenta in na svoj relativni naslov glede na začetek segmenta. Vsak program je tu videti kot zbirka med seboj povezanih segmentov. Pri prehajanju iz enega segmenta v drugega so običajno predpisane omejitve. Segment ne more dostopati do podatkov v drugem tako, kot dostopa do lastnih. To lahko naredi le na predpisan način, ki ima tudi veliko vrsto preverjanj. To pomeni, da lahko na vsak segment gledamo kot na svoj pomnilniški prostor, ki je sicer povezan z drugimi, ampak je od njih neodvisen. Zato se segmentacija označuje tudi kot večdimenzionalni navidezni pomnilnik. Vsak segment ima svoje simbolično ime – simbolična segmentacija.
Največja velikost segmenta in največje število le-teh sta zaradi praktičnih razlogov omejena.
Preslikovalna funkcija je realizirana preko tabele segmentov. Velikost tabele ni znana vnaprej. Običajno ima lahko vsak program svojo tabelo segmentov. Vsak segment ima v tabeli svoj deskriptor. Kot pri ostranjevanju spodnjih s bitov določa naslov besede znotraj segmenta zgornjih n-s bitov pa številko segmenta. Do naslova segmenta pridemo tako, da številko segmenta seštejemo z vsebino v registru tabele segmenta (naslov tabele) in tako pridemo do naslova segmenta. K temu naslovu se prišteje še relativni naslov besede (ne doda kot pri ostranjevanju).
Parametri v deskriptorju so podobni kot pri ostranjevanju:

- prisotni bit P: če je segment v glavnem pomnilniku je 1
- zaščitni ključ RWX: enako kot pri o., vendar je zaščita popolnejša, ker vemo kaj je v segmentu. Določene dele programa lahko zaščitimo tudi pred samim seboj.
- umazani bit C se postavi na 1, če je prišlo do spreminjanja segmenta medtem, ko je bil v glavnem pomnilniku.
- velikost segmenta L vsebuje trenutno velikost merjeno v številu besed. Največja velikost je 2^s besed, zato mora biti parameter L dolg s bitov. Pri vsakem dostopu se s-bitni naslov znotraj segmenta primerja z L, če je večji imamo napako – sproži se past.
- naslov segmenta SA vsebuje fizični naslov segmenta v glavnem pomnilniku.
- lahko imamo tudi deskriptor G, ki pove ali je segment globalen ali lokalen.

Segmente vseh programov, ki se trenutno izvajajo imenujemo obstoječi segmenti. Običajno imamo v okviru OS tabelo obstoječih segmentov. Več programov lahko uproblja isti segment.
Vsak segment se mora prenesti v zadosti velik zvezen prostor. V večini primerov, ta prostor ne bo točno tako velik kot segment. V pomnilniku bodo nastale luknje, katerih vsota je lahko velika za nov segment. Temu pojavu pravimo ZUNANJA FRAGMENTACIJA. Problem je v tem, da pri notranji fragmentacija poznamo njeno povprečno in tudi največjo velikost, zunanja pa se stalno spreminja in jo je zato težko nadzorovati. Imamo pa tri algoritme, s katerimi poskušamo rešiti problem zunanje fragmentacije:

- najboljše ujemanje – poiščemo najmanjšo luknjo v katero bi pasal naš segment
- najslabše ujemanje – poiščemo največjo luknjo in vanjo damo segment

- prvo ujemanje – segment damo v prvo luknjo, ki je zadosti velika

Tretja možnost je **segmentacija z ostranjevanjem**. Z njo želimo izkoristiti prednosti segmentacije, vendar brez zunanje fragmentacije. Vsak segment razdelimo na strani, kar imenujemo linearna segmentacija. Ta rešitev je boljša, saj ni več potrebno, da je glavnem pomnilniku vedno cel segment – v njem so samo tiste strani, ki se trenutno uporabljajo.
Glavni pomnilnik je razdeljen na okvire strani. Navidezni naslov je sestavljen iz treh delov: številke segmenta, številke strani in naslova besede znotraj strani. Najprej se seštejeta vsebina registra tabele segmenta (naslov tabele segmenta) in (n-s) bitna številka segmenta iz virtualnega naslova. V deskriptorju segmenta se vzame naslov tabele strani (PTA) in se sešteje s (s-p) bitno številko strani v segmentu. Poleg tega se preveri, če je številka strani ta številka primerja z dolžino segmenta, ki je v kontrolnih bitih deskriptorja segmenta (bit L). Če je Številka večja, se sproži past. Sicer dostopamo do elementa tabele strani segmenta, do deskriptorja naše strani v segmentu, kjer je poleg kontrolnih bitov tudi FN (številka okvirja). To številko pomnožimo z 2^p in jo prištejemo zadnjim p bitom iz navideznega naslova – naslovu besede znotraj strani.

46) Notranja in zunanja fragmentacija

Glej 45.

47) Kako pospešiti preslikovanje

Pri vsakem dostopu (tudi če podatek ni v navideznem pomnilniku) je potrebno narediti preslikavo navideznega naslova v fizični naslov. To pomeni, da sta pri vsakem pomnilniškem dostopu potrebna najmanj dva dostopa: dostop do tabele strani v glavnem pomnilniku in dostop do fizičnega naslova kjerkoli že je. Če imamo segmentacijo z ostranjevanjem ali večnivojske tabele se to poveča na 3/4 dostope. Zaradi tega imajo računalniki z navideznim pomnilnikom v CPE vgrajen mehanizem, ki skrajša čas za preslikovanje – majhen predpomnilnik, ki vsebuje nazadnje uporabljene deskriptorje. Pravimo mu **preslikovalni predpomnilnik** (TLB). V njem so vedno deskriptorji. Dolžina bloka je običajno enaka dolžini deskriptorja.
Pri segmentaciji z ostranjevanjem imamo običajno dva TLB-ja. Eden vsebuje deskriptorje segmentov, eden pa deskriptorje strani. Če je preslikava večnivojska je dovolj, da se v TLB prenese samo deskriptor najnižjega nivoja. TLB ima naslednje lastnosti: velikost 32 do 2048 deskriptorjev, 1 do 2 deskriptorja na blok, 0,5 do 1 urina perioda za dostop, 10 do 100 period zgrešitvene kazni in od 99 do 99,9% verjetnost zadetka. Dostop do TLB in preslikovanje tečeta paralelno z dostopom do predpomnilnika. Če imamo ločen operandni in ukazni predpomnilnik moramo imeti tudi dva TLB-ja.

48) Serviranje napake strani / segmenta

Zgrešitev v TLB pomeni eno od dveh možnosti:

- stran ali segment je v glavnem pomnilniku, v TLB se prenese nov deskriptor in naredi se dostop
- strani ali segmenta ni v glavnem pomnilniku. Sproži se past in operacijski sistem poskrbi za prenos strani/segmenta z diska in za spremembo deskriptorja te strani v tabelo strani

Če je bil bit P enak 1, vemo da se je zgodila prva možnost, sicer druga. Ukaz, ki ga trenutno izvajamo moramo prekiniti s pastjo. Imamo dve možnosti. Lahko obnovimo stanje, lahko pa ponovimo ukaz, s tem da moramo paziti da resetiramo vse kot je bilo prej. Na RISCih uporabljamo ponovitev, na CISCih pa je na nekaterih ukazih celo ne smemo uporabiti, ker bi bilo potem delovanje napačno. To je mogoče rešiti tako, da CPE pri tistih ukazih, ki bi pri ponovitvi povzročili napako preveri, če bo prišlo do napake strani.

Servisni program mora najprej shraniti stanje programa, nato mora ugotoviti navidezni naslov na katerem je prišlo do napake. Če je bila napaka pri ukazu, je ta naslov shranjen v PC, če je bila pri operandu se lahko določi z analizo ukaza. Ko pozna naslov z uporabo številke strani, ki je v njem dostopa do tabele (strani) in dobi naslov te strani na disku. Določi kateri od okvirov v glavnem pomnilniku se bo zamenjal. Če je umazani bit v tem okviru postavljen, se mora shraniti nazaj na disk. Stran se prenese z diska nazaj v izbrani okvir. Vsebina deskriptorja v tabeli strani se spremeni.

V tem času (več deset milijonov urinih period) CPE ne stoji. Izvajati se začne nek drug program. Ko je prvoten program spet sposoben za nadaljevanje se lahko začne izvajati naprej, lahko pa počaka da pride do napake v trenutnem programu.

49) Načini za zmanjševanje prostora za tabele strani/segmentov

Prostor, ki ga zasedajo tabele ni mogoče uporabiti za programe – prostor je neizkoriščen (okoli 64MB). Potrebujemo pametnejšo organizacijo tabel, in sicer:

- imamo lahko **večnivojske tabele strani**, katerih vsak del je lahko v svojem delu pomnilnika. Tudi če bi bili vsi njeni deli v glavnem pomnilniku, bi bila izkoriščenost veliko večja. DEC, MOTOROLA, 80X86
- uporaba **invertirane tabele strani**, kar pomeni da imamo v tabeli samo tiste strani, ki so trenutno v glavnem pomnilniku. Problem je v tem, da iz številke strani ni razvidno, ali in kje stran v glavnem pomnilniku je, zato moramo uporabit razpršilno funkcijo. Če za neko številko strani preslikava v tabelo obstaja, se dobljena številka uporabi kot številka okvira, sicer je napaka. Power PC.
- tabele lahko shranjujemo v **navideznem pomnilniškem prostoru**. Če zagotovimo, da so nekateri programi in nekatere tabele vedno v glavnem pomnilniku (del OS), je

večina tabel strani lahko v navideznem pomnilniku. Ta rešitev se uporablja skupaj z večnivojskimi tabelami. Število napak strani se sicer nekoliko poveča, vendar se ta rešitev vseeno veliko uporablja.

50) Izbira velikosti strani

Pri čisti segmentaciji se to vprašanje ne pojavlja. Drugače pa je vprašanje podobno vprašanju kako velik naj bo predpomnilniški blok.
Na odločitev o velikosti strani vplivajo:

- **velikost sektorja na magnetnem disku**: pri prenosu z diska se vedno prenese najmanj en sektor (512 bajtov). To pomeni, da je izkoristek najboljši, če je velikost strani večkratnik velikosti sektorja.
- **izkoriščenost pomnilnika**: dele pomnilnika uporabnik nikoli ne more uporabiti za svoje programe. Med njih spada notranjo fragmentacijo, zaradi katere bo del pomnilnika v velikosti polovice strani neizkoriščen. Z zmanjševanjem strani se zmanjšuje tudi notranja fragmentacija, vendar se istočasno povečuje velikost tabel. Torej bo v povprečju pomnilniški prostor, ki ga ni mogoče uporabiti odvisen od velikosti segmenta (programa). Vzemimo da je velikost segmenta Ss, velikost strani pa Sp. Pri vsakem segmentu je torej neizkoriščenih N besed: N=Sp/2 + c Ss/Sp, pri čemer je c velikost deksriptorja. Prvi člen predstavlja notranjo fragmentacijo, drugi pa prostor, ki ga zaseda tabela strani. Torej lahko definiramo izkoristek segmenta: u=Ss/(N+Ss)=2SsSp/(Sp^2+2Ss(c+Sp)). Odvajamo po Sp, pri maksimumu mora biti odvod enak 0. Iz tega dobimo, da mora biti optimalna velikost strani enaka **√2cSs**. Ekstrem je zelo neizrazit, zato lahko kar precej odstopimo od njega, pa bo izkoristek še vedno dokaj optimalen.
- **število napak strani** je najpomembnejši parameter zaradi premetavanja – delovanje računalnika se lahko zreducira na menjanje strani v fizičnem naslovnem prostoru. Lahko ga samo ocenimo z analizo dogajanja med delovanjem računalnika. Opazujemo dva zaporedna naslova A(i) in A(i+1). Imamo tri možnosti:

1. če sta oba naslova ukazov in je A(i+1)=A(i)+d, je d v večini primerov zelo majhen
2. če sta oba naslova operandov je analogno 1.
3. če je A(i) ukaz, A(i+1) pa operand ali obratno je d velika številka. Razlog je v načinu programiranja – podatki so na enem koncu ukazi pa na drugem.
   Če se stran povečuje zajame prvo točko, verjetnost zgrešitve za ta dva primera bo večja, obenem pa bo v tretji točko zgrešitev naraščala, saj se število strani zmanjšuje, kar pa zajame manjšo časovno lokalnost. Torej bo do neke mere povečevanje strani dobro, potem pa se bodo zgrešitve začele pojavljati pogosteje. Najpomembneje je torej, da čimbolj zmanjšamo število napak strani.

Omeniti moramo še to, da je čas za prenos večjega bloka zanemarljivo večji od časa za prenos manjšega (pri predpomnilniku temu ni tako).

51) Strategije pri uporabi navideznega pomnilnika

Operacijski sistem igra pri navideznem pomnilniku pomembno vlogo. Omogočiti mora, da se dana množica programov izvrši čim hitreje. Ubadati se mora z naslednjimi problemi:

- koliko okvirov strani v glavnem pomnilniku naj ima nek program
- kdaj, katere in koliko strani naj se prenese iz pomožnega v glavni pomnilnik
- katere strani naj se prenesejo iz glavnega nazaj v pomožni pomnilnik

Ta pravila imenujemo **dodeljevalna, polnilna in zamenjevalna** strategija. Strategije se nekoliko razlikuje od tistih pri predpomnilniku, saj je tam osnovni cilj povečati verjetnost zadetka, tu pa moramo povečati izkoriščenost računalnika. Poleg tega se pri navideznem pomnilniku uporabljajo programsko realizirane strategije, saj je časa veliko več. Označujemo jih kot strategije za upravljanje s pomnilnikom.

V danem trenutku je stopnja multiprogramiranja d (število aktivnih programov). Vsakemu programu pripada določeno število okvirov v glavnem pomnilniku – okviri so v t.i. rezidenčni množici. Pri uporabi dodeljevanje strategije fiksna razdelitev je število okvirov, ki so dodeljeni programu fiksno in se med izvajanjem ne spreminja. Boljši način je spremenljiva razdelitev, ki ta prostor dinamično spreminja. Poleg tega lahko razdelitev razdelimo še na globalno in lokalno. Medtem ko lokalna pri odločanju upošteva samo trenutne lastnosti rezidenčne množice programa, v katerem je prišlo do napake, globalna upošteva zgodovino rezidenčnih množic vseh programov.
Pri polnjenju okvirov imamo dve strategiji: polnjenje na zahtevo prenese stran v okvir, ko pride do zgrešitve (kot pri predpomnilniku), vnaprejšnje polnjenje pa prenese poleg nje še sosednje strani, za katere predvideva, da se bodo v bližnji prihodnosti uporabljali.
Pri izbiri ustreznega algoritma moramo najprej analizirati dogajanje med izvajanjem programa. Stopnja lokalnosti ni konstantna, ampak jo lahko razdelimo na faze, v katerih se delovna množica spreminja počasi, saj je stopnja lokalnosti visoka in na prehode, za katere je značilna hitra sprememba delovne množice. Ker je čas v fazah veliko daljši, kot prehodih, predstavlja število napak v prehodih njihovo večino. Torej je vnaprejšnje polnjenje boljše.
Za napovedovanje kako algoritmi za dodeljevanje, polnjenje in zamenjevanje vplivajo na število napak strani je koristna funkcija frekvence napak strani, ki podaja število napak strani na enoto navideznega časa (čas, ki teče samo takrat, ko ni zamenjevanja strani). Ugotovimo lahko, da bo najboljši agoritem tisti, ki da najmanjšo vrednost f za določeno velikost. Namesto funkcije f se veliko uporablja tudi t.i. življenjska funkcija e, ki podaja povprečen navidezen čas med dvema napakama strani za poljuben program in je enaka 1/f.

Poseben primer pri dodeljevanju pomnilnika je izbira stopnje multiprogramiranja d. Če je d premajhen je računalnik neizkoriščen, če je prevelik se število okvirov za program zmanjša in število napak strani se zato

močno poveča. Problem izbire pravega d je znan kot problem optimalnega obremenjevanja.
Če je L(d) povprečni navidezni čas med dvema napakama strani, pri stopnji d in je L(d) = tb (tb – čas za prenos ene strani), pride do premetavanja. Naslednja napaka se zgodi še preden je bila servisirana prejšnja. Ta pojav je odvisen od velikosti glavnega pomnilnika M. Pri velikem M je področje stopnje d lahko veliko večje kot pri majhnem . Povečevanje glavnega pomnilnika je smiselno do velikosti, ko se izkoristek približa 1. Izkoristek računalnika bo največji, če se bo d prilagajala spremembam v trajanju povprečnega časa med dvema napakama tako, da je čas L(d) nekoliko večji kot tb – **L=tb kriterij**.

52) Algoritmi za optimalno uporabo navideznega pomnilnika

Algoritem mora upoštevati ugotovitve iz točke 52. Najpreprosteje je narediti dober polnilni algoritem: stran pri kateri je prišlo do napake, je potrebno vedno prenesti v glavni pomnilnik, lahko tudi sosednje. Dodeljevalne in zamenjevalne strategije s fiksno dodelitvijo in lokalnim delovanjem se zaradi slabosti ne uporabljajo.
Zagotoviti moramo da se stopnja multiprogramiranja d dinamično prilaga spremembam lokalnosti. Ko je visoka, ima lahko program malo okvirov strani in ni veliko napak. Med prehodi med fazami potrebuje program za isto število napak veliko okvirov strani. Pri d aktivnih programih je verjetnost velika, da zmanjšanje stopnje lokalnosti nekaterih programov sovpada z zvečanjem drugih. Torej mora algoritem delovati tako, da vzame okvire programom, ki imajo visoko lokalnost in jo da tistim z nizko. Algoritem mora čimbolj uspešno preprečiti premetavanje – z zmanjšanjem d, a vendar d ne sme biti prenizka, ker zaradi tega trpi izkoristek računalnika. Dva znana algoritma za dodeljevanje in zamenjevanje vse to upoštevata (pri polnjenju je rešitev preprosta – OBL algoritem, ki se uporablja v kombinaciji s sledečima):

- WS (Working Set) algoritem uporablja pojem delovne množice strani W(t,T), ki je definirana kot množica tistih strani, ki jih je program naslovil v časovnem intervalu (t-T,t), pri čemer je t navidezni čas, T pa parameter. Sestavljajo jo številke strani. Algoritem se aktivira za vsak program, ko je intervala konec. Ob prvem aktiviranju dodeli programu fiksno število strani. Kasneje dodeljuje programu toliko strani, kot je okvirov v delovni množici. Če je razpoložljivo število okvirov premajhno, se program izloči iz množice aktivnih (d–d -1). Ko število prostih okvirov preseže določeno vrednost se d poveča. Torej algoritem upošteva spremembe v lokalnosti, zato daje zelo dobre rezultate, vendar je njegova realizacija nerodna, saj ga je treba aktivirati vsakič, ko za nek program preteče čas T; za vsak program teče T drugače.
- Aktiviranje poleg tega običajno ne bo sovpadlo z napako strani. Močno bi poenostavili programiranje, če bi algoritem zagnali ob napakah strani, ko je prenos vedno potreben. Algoritem, ki deluje na ta način je PFF (Page Fault Frequency), ki je v bistvu

poenostavljen WS algoritem. Še vedno imamo parameter T, vendar se algoritem aktivira samo ob napakah. Ob vsaki napaki strani se v glavni pomnilnik prenese potrebna stran. Število okvirov programa se poveča za 1. Če pri tem zmanjka prostih okvirov, se program izloči iz množice aktivnih. Potem se navidezni čas od prejšnjega zagona algoritma primerja s trenutnim časom in če je manjši od T algoritem ne naredi ničesar drugega, sicer se programu dodeli toliko okvirov strani, kot jih je v delovni množici programa. Vrednost 1/T lahko razumemo kot največjo frekvenco napak strani. Če je frekvenca večja, se število okvirov prilagaja delovni množici ob vsaki napaki. T služi kot varnostni prag, da se število okvirov ne bi prehitro zmanjševalo.